龙芯ejtag简明手册

龙芯中科 乔崇

September 28, 2014

Contents

Abstract

EJTAG是mips的onchip debug调试标准。现在龙芯1号、龙芯2号(龙芯2F和以前版本不支持)和龙芯3号系列都支持ejtag调试。通过ejtag可以大大方便软件调试,这里讲讲ejtag原理和ejtag-debug软件。
ejtag-debug是我编写的一个ejtag调试调试工具，支持读写寄存器、内存、反汇编、执行用户编写的小程序、gdb远程调试和脚本语言。

1 安装

软件下载地址为http://www.loongson.cn/uploadfile/embed/ls1b/ejtag/。
其中ejtag-debug-xxx.tar.gz是linux版本。ejtag-debug-cygwin-xxx.rar为windows版本。 请下载安装最新版本使用。

1.1 windows下安装

- 龙芯ejtag在windows下使用需要安装驱动，如果windows下没有安装驱动执行ejtag_debug_usb.exe会提示缺少libusb库，或者找不到设备。驱动按照过程如下:

  1. 首先将usb电缆插入pc机的usb口，这个时候windows会检测出未知的usb设备插入提示按照驱动
  2. 选择手动安装，并指定安装目录为ejtag-debug/driver目录,然后下一步来自动安装驱动
  3. 还可以在设备管理器里面找为未安装驱动的usb设备，vid: 2961,pid: 6688，然后指定按照ejtag-debug/driver目录里面的驱动即可。

  驱动安装完后，直接双击ejtag_debug_usb.exe执行就可以了。

- ejtag程序的一些命令会调到脚本，需要安装perl。可以下载active perl或者strawberry perl都可以http://www.perl.org/get.html#win32.
- 如果在cygwin下执行，需要将ejtag-debug里面的cygwin1.dll删除，否则程序会自动退出去。

1.2 Linux下安装

在linux下不需要安装驱动，直接以超级用户权限执行ejtag_debug_usb.exe即可。

1.3 配置文件和启动参数

配置文件是ejtag.cfg 程序会自动打开./ejtag.cfg并执行里面的内容。 因为本软件同时支持龙芯1、2、3三个系列的处理器，在运行之前需要根据你要调试的处理器类型对ejtag.cfg进行配置。

- setconfig core.cpucount 1 设置处理器核的个数，如龙芯3A有4个处理器核，应该设置成4，龙芯3B设置成8，龙芯2H、龙芯1A、1B、1C、1D设置成1。
- setconfig core.cpuwidth 32 设置处理器寄存器宽度，如龙芯3A是64位处理器，应该设置成64，龙芯3B、龙芯2H设置成64，龙芯1A、1B、1C、1D设置成32。
- usblooptest设置usb ejtag的分频和位相选择,当其他都配置正确ejtag还不正常工作时，可以尝试对ejtag时钟降频

```
usblooptest help
usblooptest 0x81000070 0x10000|divison #TCLK freq divided by division, divison must only one bit
    set
usblooptest 0x81000070 0x20000|samplesel #sample tdi output at the samplesel clk after rising edge
     of tck, samplesel value from 0 to 3
```

- selectcore coreid
  selectcore命令用于龙芯3b/3c选择多核ejtag的串接方式0-7表示调试单个核，-1表示8个核串到一起。

其他和功能相关的一些设置:

- helpaddr: 帮助地址，帮助程序执行的地址，地址应该指向ddr,cache/uncache都可以
- usb_ejtag.put_speed 范围0 0xffff，调节ejtag的usb读取速度,设置越大速度越慢,当主机速度较慢的时候需要配置延迟大一些
- usb.maxtimeout usb访问的最大超时时间，如果usb_ejtag.put_speed设置较大，usb.maxtimeout也要设置更大一些
- putelf.uncached: 设置0 putelf、program命令上传数据到cache地址，然后刷cache, 1在put、program先刷cache然后上传到uncache地址,2put、program上传到uncache地址，不刷cache. 如果设置为0，1一定要保证cache被初始化，如果cache没被初始化运行cache_config和cache_int命令

- timer 1000:设置timer 1000ms检查一次ejtag状态，如果usb速度慢设置更大一些，timer 0关闭timer检测ejtag状态功能

1.4 命令行参数

./ejtag_debug_usb

1.5 ejtag-debug参数

```
./ejtag_debug_usb [-dlStch] [-e cmd] [-T n]
-d: verbose on,show debug messags
-e 'cmd': run cmd
-l: do not use read line
-S: log disassemble info
-s: run cmdserver
-t: disable timer
-T n: set timer n ms
-c: do not load cfg file
-h: show this help
```

1.6 ejtag使用顺序

1. ejtag.cfg里面要设置正确core.cpucount,core.cpuwidth
2. 主板先上电
3. ejtag usb端插入
4. ejtag 插头插到主板的ejtag座上，注意三角形(1脚)对主板上的三脚形(1脚)
5. 运行sudo ./ejtag_debug_usb -t来执行ejtag软件
6. 如果ejtag已经插在座上，处理器下电了，处理器再上电后需要在ejtag软件里面允许

```
jtagled trst:0 trst:1
```

来对处理器发出一个trst复位操作

1.7 ejtag连接和速度问题

- 将ejtag插入usb口可以观察到usb ejtag的两个灯开始都亮，1s后只有一个灯亮。这说明usb ejtag硬件没有问题。
- sudo ./ejtag_debug_usb -t运行ejtag软件，jtagled 1命令灯亮，jtagled 0命令灯灭。说明usb驱动安装正常。
- 然后将ejtag插头插到主办上，注意1脚对1脚,ejtag.cfg里面的cache_config注释掉
- 运行sudo ./ejtag_debug_usb -t, 运行jtagregs d8 1 1来读处理器的ejtag id寄存器，如果是0x20010819或者是0x5a5a5a5a都说明连接正确
- 运行set命令读处理器的通用寄存器，如果能读出来且非全0，则说明处理器运行起来了，ejtag也连接上了
- 如果读不出来，按ctrl-c退出。可能是处理器在无程序的情况下运行到地址空洞，设备没响应，总线卡住了。可以运行resetcpu命令来复位cpu，然后按set就能读出通用寄存器内容了。
- 在进行put, get, program命令的时候，如果是虚拟机，虚拟usb速度很慢，响应的龙芯处理器要等更长时间接usb数据，setconfig usb_ejtag.put_speed要设置更大一下，同setconfig usb.maxtimeout也要设置更大一些。
- 在进行put, get, program命令的时候，可能会刷cache，一定要保证cache已经初始化过了，或者设置设置setconfig putelf.uncached 2，或者运行cache_config, cache_init先对cache进行初始化。

2 寄存器读写

2.1 通用寄存器读写

- 读通用寄存器: set [寄存器名]
- 写通用寄存器: set [寄存器名] [数值]
- save [file] : 保存通用寄存器内容到文件/临时内存中
- restore [file]: 恢复通用寄存器内容来自于文件/临时内存中

为了方面脚本软件调用，也可以用下面的方法访问

```
cpuregs
d4 1 2
m4 1 0x100或者

cpuregs d4 1 2
cpuregs m4 1 0x100
```

cpusregs表示设置d1、d2、d4、d8为1、2、4、8字节寄存器读功能，m1、m2、m4、m8为1、2、4、8字节寄存器写功能。
d4 1 2表示读寄存器1开始的两个寄存器也就是at和v0寄存器。 m4 1 0x100描述写寄存器1为0x100，也就是设置at寄存器为0x100。

2.2 协处理器读写

- 选择协处理器组为sel，默认为0: cp0regs [sel]
- 协处理器读: d4 regno [count] 或者 cp0regs sel d4 regno [count]
- 写处理器写: m4 regno value 或者 cp0regs sel m4 regno value

例子

```
读cp0_config0: cp0regs 0 d4 16 1
读cp0_config1: cp0regs 1 d4 16 1
写为cp0_config02: cp0regs 0 m4 16 2
```

3 ejtag寄存器读写

- 选择ejtag寄存器读写功能: jtagregs
- ejtag寄存器读: d4 regno [count] 或者 jtagregs d4 regno [count]
- ejtag寄存器写: m4 regno value 或者 jtagregs d4 regno value

例子

```
读ejtag id寄存器: jtagregs d4 1 1
写ejtag control寄存器为0x1000: jtagregs m4 10 0x1000
```

4 内存读写

4.1 读写内存

- 选择内存读写功能: mems
- 读内存: d4 addr [count] 或者 mems d4 addr [count]
- 写内存: m4 addr value 或者 mems m4 addr value

- 反汇编: disas addr [count] :反汇编addr开始count个指令
- [s]memcpy[1/2/4] src dst size :拷贝内存从src到dst,命令包括smemcpy1,fmemcpy1,memcpy1等等。smemcpy[1/2/4]不用helpaddr速度慢,memcpy[1/2/4]跳到helpaddr中执行速度快。
- [s]memset[1/2/4] addr c size : 设置size大小内存内容为c,命令包括 smemset1,fmemset1,memset1等等。smemset[1/2/4]不用helpaddr速度慢,memset[1/2/4]跳到helpaddr中执行速度快。

例子

```
读内存地址0xbfc00000 0x100个4字节: mems d4 0xbfc00000 0x100
写内存地址0xa0000000为0x1000: mems m4 0xa0000000 0x1000
```

4.2 内存测试

- 测试内存从startaddr到endaddr读写正确性，方式是写0，写-1然后读回看是否相等: memtest startaddr endaddr
- 测试内存从startaddr到endaddr读写正确性，方式是从startaddr到endaddr写入数值，开始是initval，每次地址数据增加incval，全写完一遍后，再读出比较正确性: memtest1 startaddr endaddr initval incval

4.3 保存内存到文件

- 直接下载address地址开始size大小内存到filename中: sget filename address size
- 通过helpaddr里面的下载程序，下载address地址开始size大小到filename中: fget filename address size
- 通过helpaddr里面的下载程序，快速下载address地址开始size大小到filename中:get filename address size

helpaddr是一端可写内存，fget和get命令会先将一小段下载程序上载到helpaddr中，然后执行这段小程序来帮助下载。

4.4 上传文件到内存

- 直接上载文件到address地址开始内存中: put filename address
- 通过helpaddr里面的下载程序，上载文件到address地址开始内存中: fput filename address size
- 通过helpaddr里面的下载程序，快速上载文件到address地址开始内存中:put filename address

helpaddr是一端可写内存，fput、put和putelf命令会先将一小段下载程序上载到helpaddr中，然后执行这段小程序来帮助下载。

4.5 上传启动elf文件

另外ejtag还支持上传elf文件要加载的内容到内存对应的命令是sputelf/fputelf/putelf elffile。

- putelf elffile
  上传elf文件到内存，地址和入口信息都从elf文件获得并设置pc,sputelf/fputelf/putelf底层分别调用sput/fput/put
- initrd initrdfile [addr]
  initrd命令上载ramdisk文件到addr，addr默认是0x84000000, initrd的内存地址和大小被保存在config设置karg.rd_start,karg.rd_size文件中。
  initrd可以这样生成

```
cd ramdisk
find . |cpio -o -H newc|gzip -c > initrd.gz
```

- karg arg0 arg1 arg2 ...
  karg命令将内核参数arg0 arg1 ...和所有ENV_开始的环境变量转化成内核参数和环境变量格式写入setconfig karg.bootparam_addr设置的地址里面(默认是0xa4000000)。karg会自动根据initrd的信息设置rd_start,rd_size内核参数。因此initrd命令要在karg命令前运行。
- pmon的环境变量xxx的数值yyy通过程序的ENV_xxx环境变量传递到内核里面，程序的环境变量可以通过setenv命令设置，也就是说通过setenv ENV_xxx yyy设置，比较重要的环境有cpuclock,memsize,highmemsize,设置方法如下

```
    setenv ENV_memsize 256
    setenv ENV_highmemsize 0
    setenv ENV_cpuclock 266000000
```

  分别是设置内核内存大小和cpu的时钟。

一般上传内核命令如下:

```
putelf vmlinux
karg console=ttyS0,115200 rdinit=/sbin/init initcall_debug=1
cont
```

5 cache相关命令

- cache_config : cache大小自动获取
- cache_init : cache初始化
- cacheflush addr size : 刷新数据和指令cache从地址addr开始，大小为size。当size大于config cacheflush.nohelp_size用helpaddr
- cache op addr size : 刷cache op操作到开始于addr,size大小内存。当size大于config cacheflush.nohelp_size用helpaddr
- cache1 op addr size : 刷cache op操作到开始于addr,size大小内存, 在helpaddr上执行速度快

6 flash烧写

6.1 配置内存参数

- 龙芯1A: source configs/config.ls1a;call configddr;
- 龙芯1B: source configs/config.ls1b;call configddr;
- 龙芯3A: source configs/config.ls3a;call configddr;
- 龙芯2H: source configs/config.ls2h;call set_ddr_pll;call configddr;

6.2 烧写flash

- 龙芯1A: source configs/config.ls1a;call configddr;call erase;call program
- 龙芯1B: source configs/config.ls1b;call configddr;call erase;call program
- 龙芯3A: source configs/config.ls3a;call configddr;call erase;call program
- 龙芯2H: source configs/config.ls2h;call set_ddr_pll;call configddr;call erase;call program

6.3 cache锁定烧flash

龙芯232,264,464 ip都支持cachelock功能，因此可以利用cachelock来做flash烧写而不使用内存。

- 龙芯1A: source configs/config.ls1a;call program_cachelock
- 龙芯1B: source configs/config.ls1b;call program_cachelock
- 龙芯3A: source configs/config.ls3a;call program_cachelock
- 龙芯2H: source configs/config.ls2h;call program_cachelock

7 调试功能

7.1 软件指令断点

- 设置软件指令断点到addr: b addr
- 删除addr上的指令断点: unb addr

软件断点触发的时候，处理器会停止进入ejtag调试状态。这个时候，当timer非0的时候，ejtag软件会自动打印出断点信息和指令内容。

7.2 硬件指令断点

- 设置软件指令断点到addr: hb addr [ibm]
- 删除addr上的指令断点: unhb addr [ibm]

硬件断点触发的时候，处理器会停止进入ejtag调试状态。这个时候，当timer非0的时候，ejtag软件会自动打印出断点信息和指令内容。
其中ibm是instruct address bit mask，硬件判断地址相等的方法是pc&~ibm == addr&~ibm，也就是ibm为1的位不比较地址。

7.3 数据断点

- 设置写数据到addr触发断点: watch addr [dbm]
- 设置读数据到addr触发断点: rwatch addr [dbm]
- 设置读写数据addr均触发断点: awatch addr [dbm]
- 删除addr的数据断点: unwatch addr

硬件断点触发的时候，处理器会停止进入ejtag调试状态。这个时候，当timer非0的时候，ejtag软件会自动打印出断点信息和指令内容。
其中dbm是data address bit mask，硬件判断地址相等的方法是load store address&~dbm == addr&~dbm，也就是ibm为1的位不比较地址。

7.4 单步调试

- 单步count次: si [count]
- 取消单步: unsi

当timer非0的时候，ejtag软件会自动打印出单步发生的时候的信息和指令内容。

8 gdb调试功能

8.1 gdbserver功能

- 启动gdbserver: gdbserver [port]

调试32位处理器

```
mipsel-gdb elffile
gdb) set architecture mips:isa32
gdb) set mips abi o32
gdb) target remote :port
```

调试64位处理器

```
mipsel-gdb elffile
gdb) set architecture mips:isa64
gdb) set mips abi n64
gdb) target remote :port
```

8.2 内嵌gdb功能

- 内嵌gdb调试程序:

```
gdb elffile
```

需要设置好core.abisize，调用的脚本是scripts/gdb.pl，需要系统安装perl

内嵌gdb将gdbserver和gdb功能集成到一起，操作方式和普通gdb完全一样。
执行的gdb命令默认运行./mipsel-gdb,可以设置环境变量GDB来设置其他gdb，
建议使用gdb-multiarch。

8.3 内嵌gdb模块调试功能

将模块ko文件拷贝到ejtag的modules目录中。

```
gdb elffile
gdb)modules
```

gdbmod执行gdb调试， gdb中执行modules命令自动load模块调试信息。

9 其他命令

- cont
  cont命令让程序推出ejtag debug状态，继续执行
- resetcpu [arg0] ...
  无参数的时候，通过写0x11000到ejtag控制寄存器来复位处理器并进入debug状态,带参数的时候将参数直接写到control 寄存器里面，如resetcpu 0x10000 0可以使处理器复位后继续执行。
- cpus [count] [file] 扫描每个处理器的asid和pc，格式是低32位是pc，高8位asid。当存文件时，只保存pc数值。这个命令不影响处理器的执行，但需要处理器支持pc sample
- sample [count] [file] 通过触发ejtag异常来获得pc。这个命令会不断中断处理器执行，处理器执行会变慢。

10 二进制扩展功能

- 在ejtag地址空间执行bin程序: scallbin bin/xxx.bin
- 在内存地址中执行bin程序, 普通加载，程序放在helpaddr开始的一段区域中: fcallbin bin/xxx.bin
- 在内存地址中执行bin程序, 快速加载，程序放在helpaddr开始的一段区域中: callbin bin/xxx.bin

在ejtag调试程序的bin目录下，集成了一个ejtag小系统，可以直接调用c库函数，printf会直接在ejtag界面打印信息。

实现memset的bin程序test.c

```
int mymain(char *buf,int len)
{
    printf("this is a test\n");
    memset(buf,c,len);
    return 0;
}
```

编译:

```
make CROSS_COMPILE=mipsel-linux- test.bin
```

调用:

```
callbin bin/test.bin 0xa0000000 0x12
```

默认打印是通过ejtag打印到pc机端，如果要打印到串口，可以定义putchar函数:

实现memset的bin程序test.c

```
int putchar(c)
{
 *(volatile char *)0xb4000000=c;
}
int mymain(char *buf,int len)
{
    printf("this is a test\n");
    memset(buf,c,len);
    return 0;
}
```

11 脚本扩展功能

11.1 集成简单脚本功能

- local定义局部变量

定义3个局部变量a b c

```
local a b c
```

- let对变量赋值,支持同时对多个变量复制,如果首先在函数内找局部变量，否则找全局变量，否这设置环境变量

定义变量a b c并分别设置为1 2 3

```
local a b c
let a b c 1 2 3
```

- unlet删除变量,可以同时删除多个变量
- letl对局部变量赋值,如果局部变量不存在创建这个局部变量，支持同时对多个变量复制
- unletl删除局部变量,可以同时删除多个局部变量

定义局部变量a b c并分别设置为1 2 3

```
letl a b c 1 2 3
```

- expr 表达式计算
- expr1 表达式计算,同shell里面的expr命令
- test 判断文件和表达式，和shell命令test一样
- $(cmd) 命令结果替换
- $(+cmd) 命令结果替换，但fork子进程
- {cmd} 等价于$(expr cmd)
- if value cmd
- while value cmd
- while循环

```
 do while value
 cmd1
 cmd2
 loop_break [n]
 loop_continue [n]
 end
```

- for循环

```
for letl i 1;$(expr $i<100);letl i $(expr $i+1)
cmd1
loop_break [n]
loop_continue [n]
end
```

- if,elsif,else

```
do if value
cmd1
elsif val1
cmd2
else
cmd3
loop_break [n]
loop_continue [n]
end
```

- function 函数

```
function fname
cmd1
cmd2
cmd3 $1 $2 $3
ret [val]
```

- call fname [arg1] [arg2...] :调用function fname
- source命令调用命令脚本: soure cmdfile
- 添加脚本成为新命令: newcmd cmdname oldcmd

简单脚本类子

```
function sum
local a i
let a i 0 0
do while $(expr $i<=$1)
let a $(expr $a+$i)
let i $(expr $i+1)
end
echo $a
ret $a

function sum1
for letl a i 0 0;$(expr $i<=$1);letl i $(expr $i+1)
let a $(expr $a+$i)
end
echo $a
ret $a

call sum 100
call sum1 100
echo $?
```

简单脚本类子

```
function fib
local i r
let i $1
do if $(expr $i==0)
let r 0
```

```
elsif $(expr $i==1)
 let r 1
else
 let r $(expr $(call fib $(expr $i-1))+$(call fib $(expr $i-2)))
end
echo $r
ret $r

function fib1
local i n r0 r1
letl i r0 r $1 0 1

do if $(expr $i==0)
let r 0
elsif $(expr $i==1)
 let r 1
else
 for letl n 2;$(expr $n<=$i);letl n $(expr $n+1)
 letl r0 r $r $(expr $r+$r0)
 end
end
expr %%lld $r
ret $r

call fib 5
echo $?
```

11.2 调用外部perl脚本

调用外部perl脚本，执行复杂命令： shell perl scripts/xxx.pl
perl脚本里面包含scripts/io.pm, scripts/io.pm是将命令封装成perl的函数。

- inb就是调用d1q
- outb就是调用m1
- inb/inh/inw/inq
- outb/outh/outw/outq
- do_cmd执行命令,直接输出到终端
- do_cmd1执行命令,结果返回

要写一个perl脚本test.pl访问ejtag，只要写一个perl程序，前面包含

```
#!/usr/bin/perl
use bignum;
push @INC,qq(./scripts);
require qq(io.pm);
```

后面就可以调用inb/outb/do_cmd等函数访问ejtag了。 调用的时候，运行:

```
    shell perl scipts/test.pl
```

test.pl

```
#!/usr/bin/perl
use bignum;
push @INC,qq(./scripts);
require qq(io.pm);
outb(0xffffffffbfe001e3,0x80);
outb(0xffffffffbfe001e0,0xd);
outb(0xffffffffbfe001e3,0x3);
printf "value is %x",inb(0xffffffffbfe001e3);
do_cmd("cont");
```

12 eclipse安装和设置

12.1 linux下安装

1. 解压eclipse-cpp-juno-SR2-linux-gtk.tar.gz，并运行eclipse即可 如果eclipse提示，没有安装gdk,继续步骤 2
2. jdk-7u21-linux-i586.tar.gz

12.2 windows下安装

到eclipse官网下载对应的eclipse-cdt发行版解压运行即可。

13 利用eclipse调试

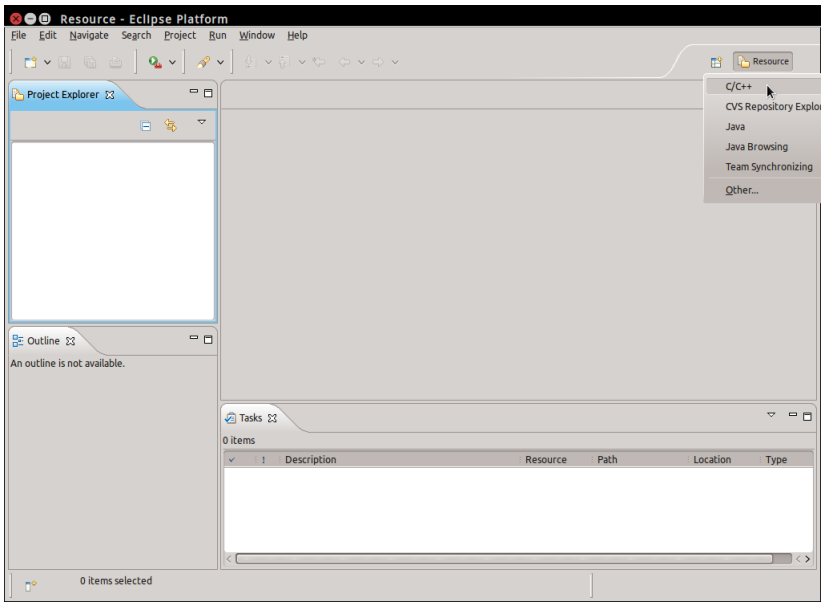

Figure 1: 选择调试c,c++

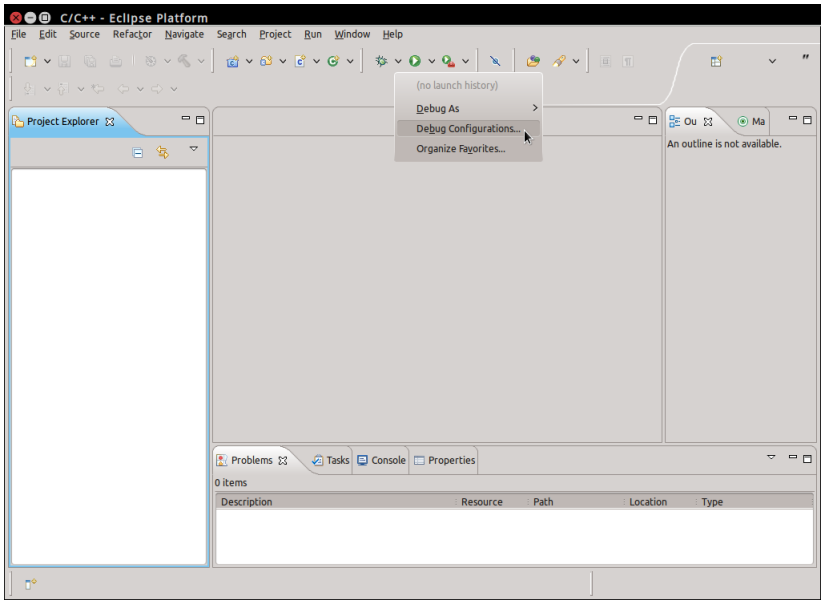

Figure 2: 选择调试c++ debug configure

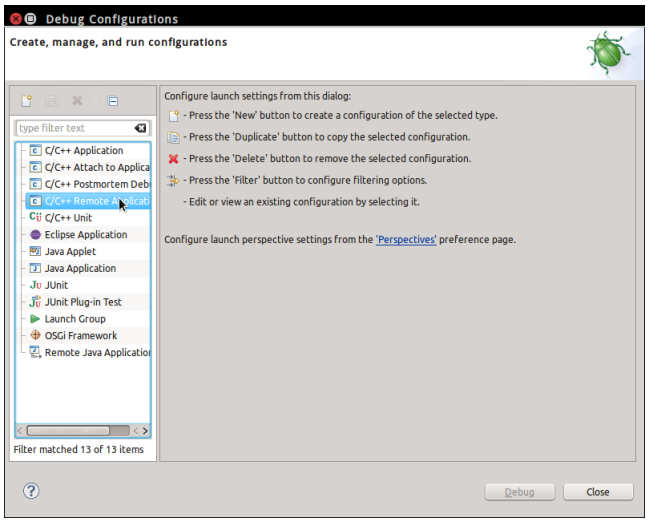

Figure 3: 选择调试c++ remote debug configure

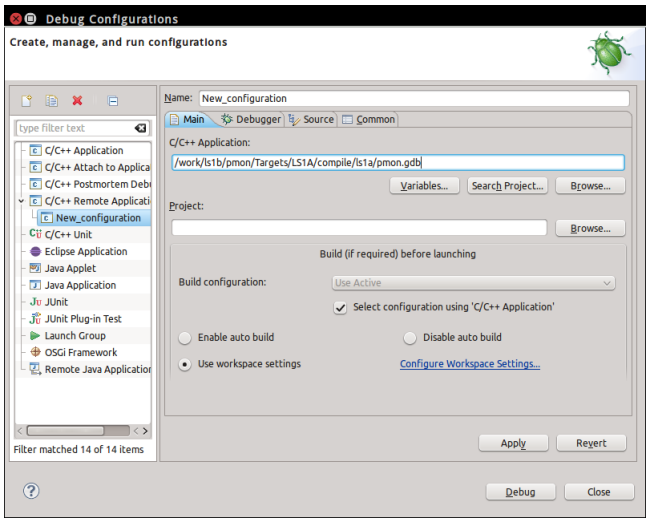

Figure 4: 选择调试c++ debug configure main

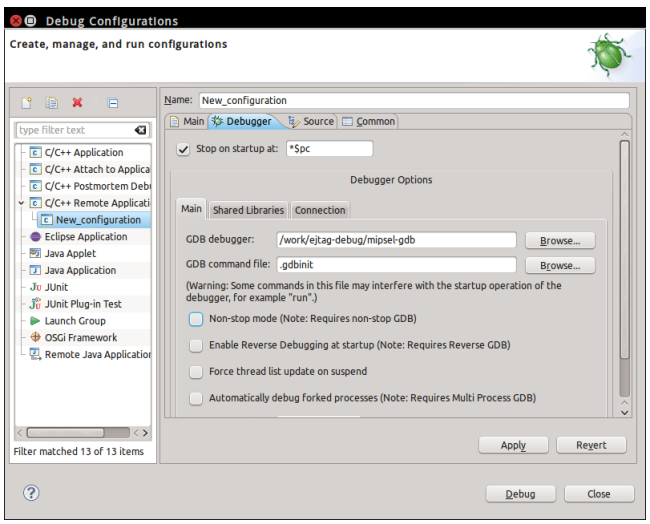

Figure 5: 选择调试c++ debugger为mipsel-gdb
break at 可以设置成*$pc，这样可以停在调试开始的地方。

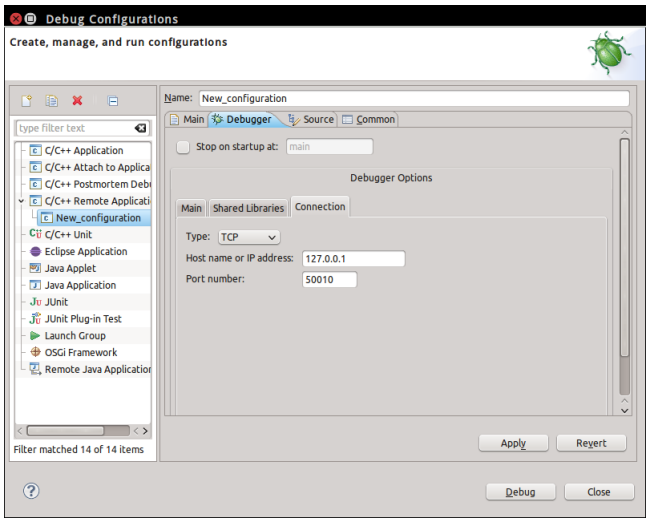

Figure 6: 选择调试c++ debug connection 127.0.0.1:50010

14 利用vxworks开发环境workbench通过ejtag来调试

- 下拉debug图标，选择debug configure

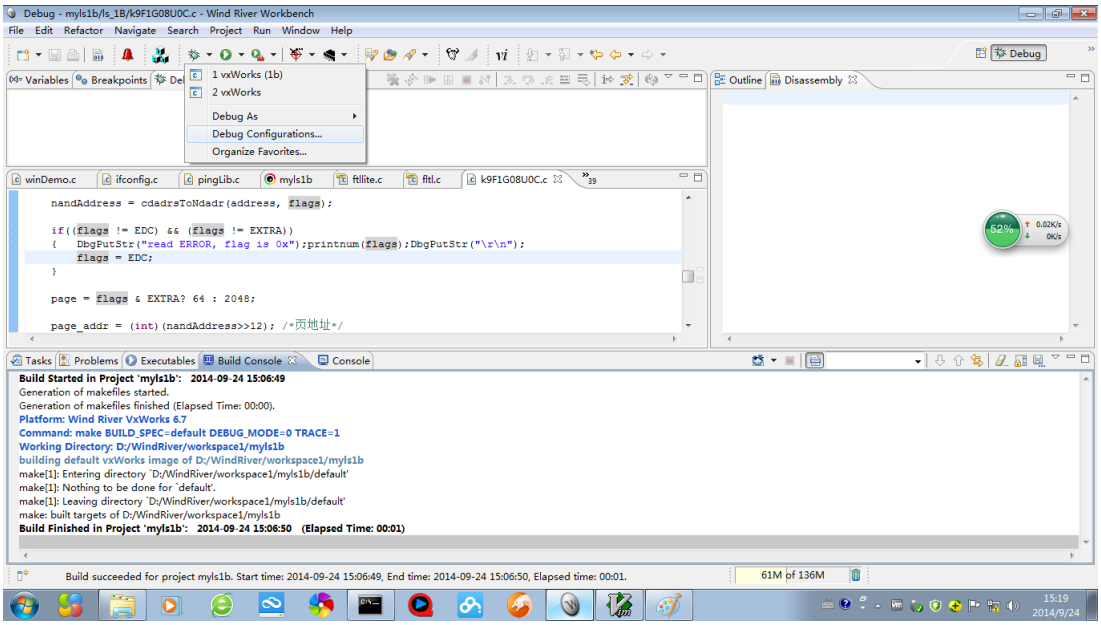

Figure 7: 选择调试c++ debug configure

- 双击 C++ local Application,新建一项, 并设置project和调试的程序

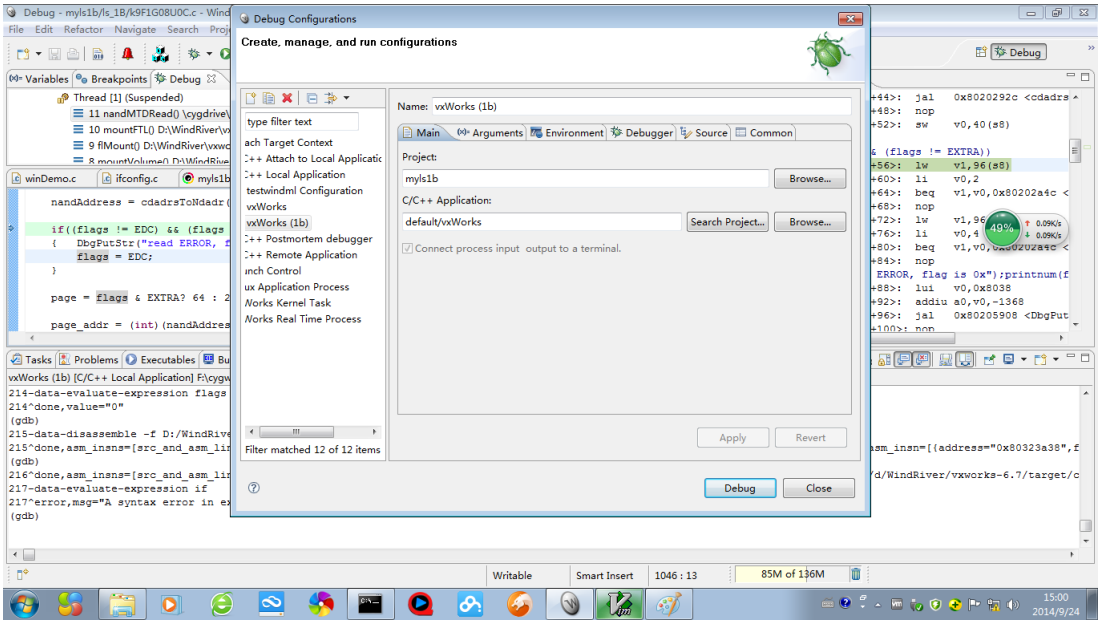

Figure 8: 双击local Application

- 设置debuger,选则gdbserver debuger,交叉gdb程序可以用ejtag程序里面的mipsel-gdb.exe

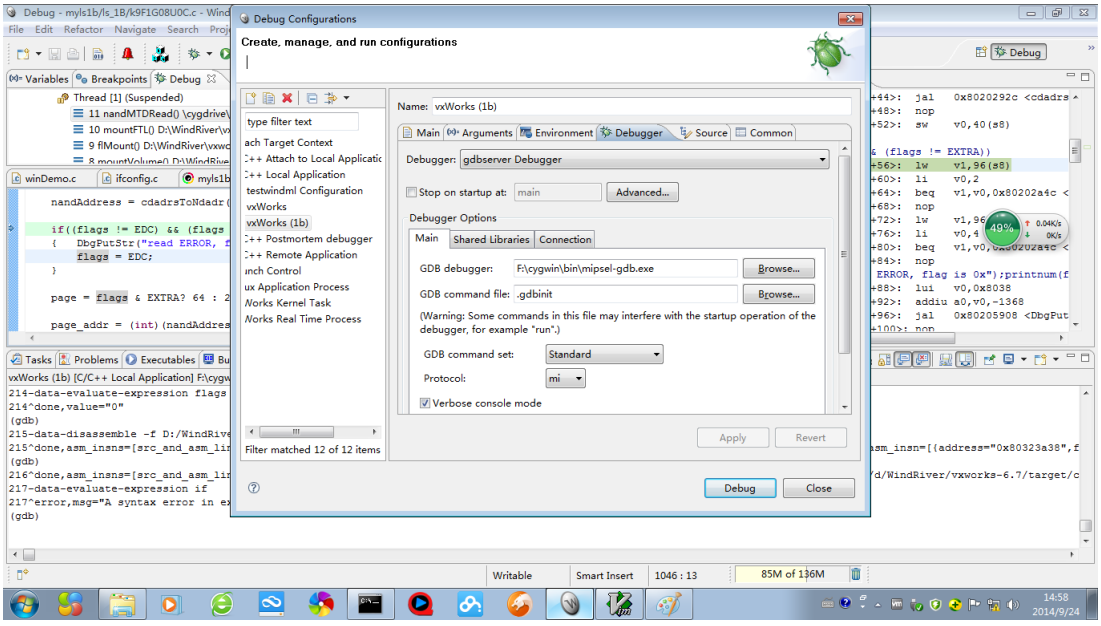

Figure 9: 设置debuger,选择gdbserver debuger

- 设置connection, server ip设置成ejtag-debug程序运行机器的ip,端口号默认设置成50010

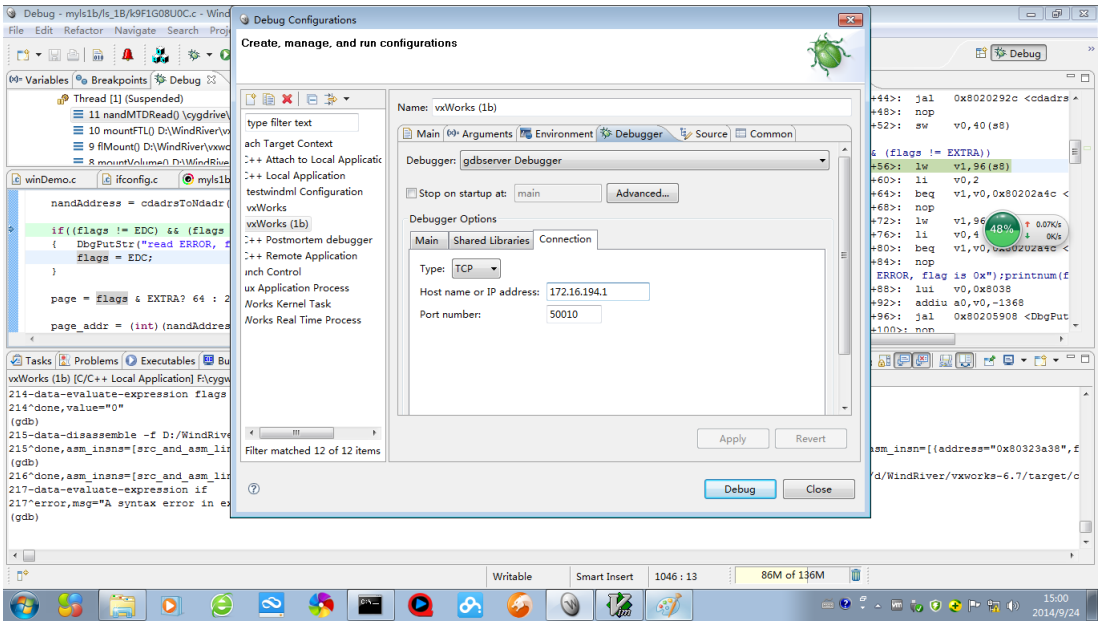

Figure 10: 设置gdbserver ip, port